DL4Seq – Ass 4

Daniel Juravski – ID 206082323
Eyal Orbach - ID 015369317

- ## Which paper you chose the implement.

We chose to implement the 'A Decomposable Attention Model for Natural Language Inference' paper.

- ## Why you chose that particular one.

We chose it because of 2 reasons:

1. The number of the free parameters of that model is very low, what was an important issue for us (and for our PC's that would run it on an appropriate time).
2. The approach of that paper was very elegant (will be detailed later), without any use of CNN or LSTM techniques, once again what prevent very expensive computationally, and having millions of parameters. They wanted to solve that problem using simple approaches (feed-forward) without any 'heavy guns' (LSTMs).

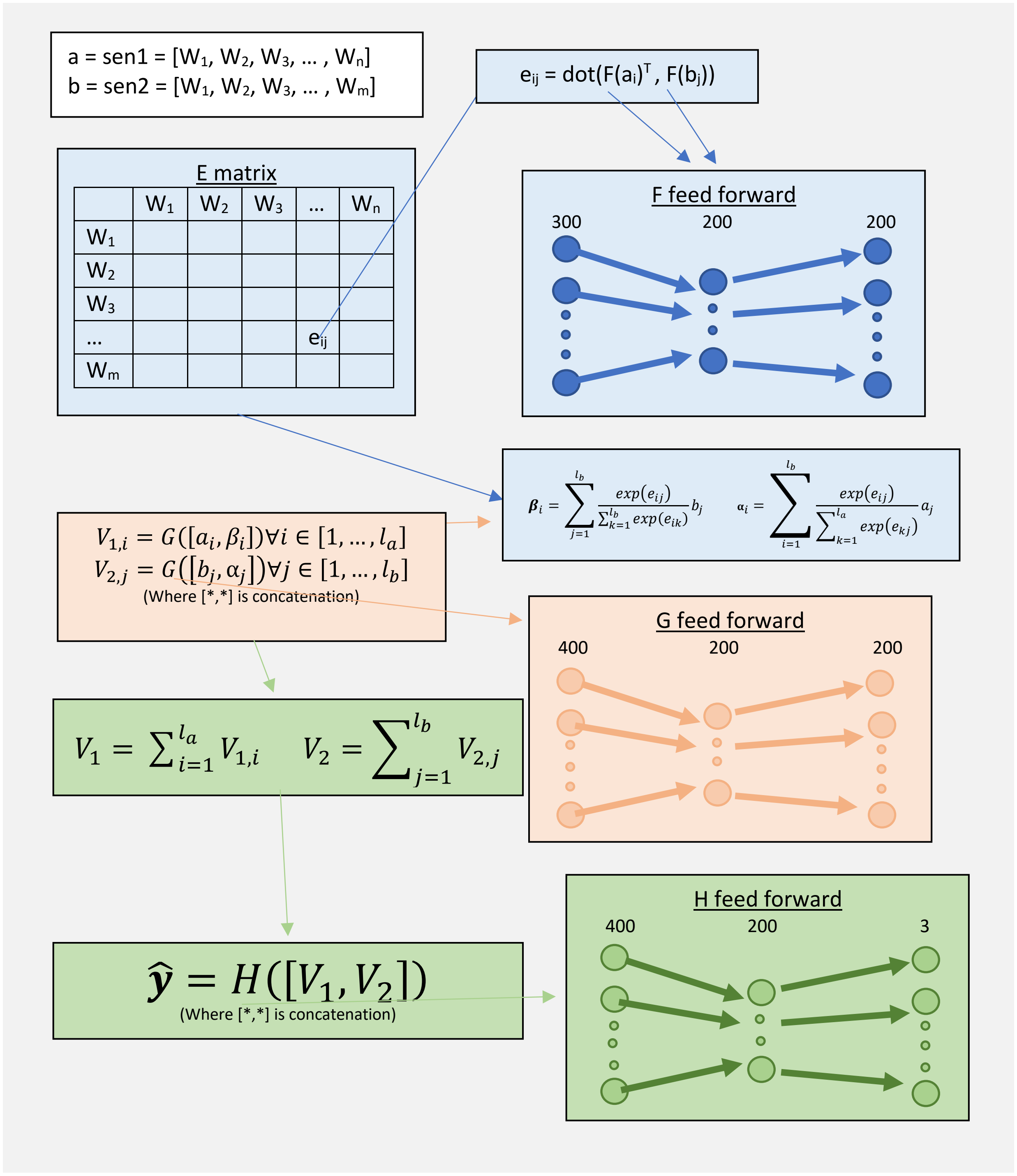
- ## What was the result reported in the paper?

	Train	**Test**
Accuracy	**89.5%**	**86.3%**

- <u>What method was used in the paper?</u>

The main method was to break the whole sentence problem into word subproblems with an emphasis on the attention principle. That was done in the following way:

1. Create a soft alignment matrix using neural attention (matrix E).
2. Decompose the task into subproblems that can are solved separately (vectors V).
3. Merge the results of these subproblems to produce the final classification.



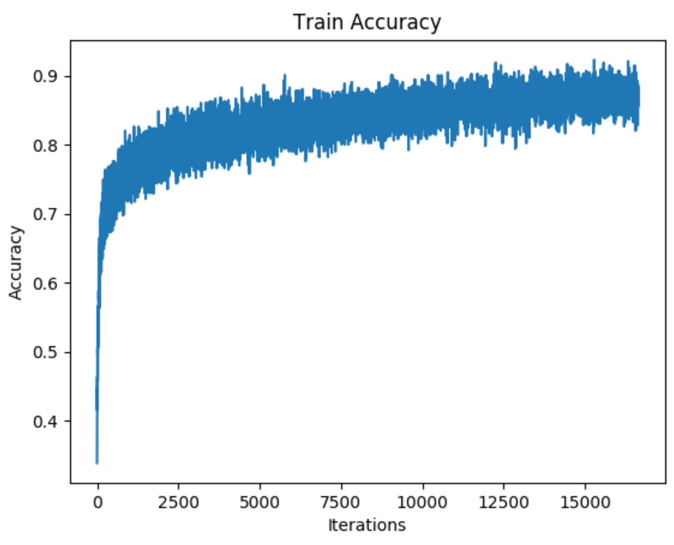
a = sen1 = $[W_1, W_2, W_3, \dots, W_n]$
b = sen2 = $[W_1, W_2, W_3, \dots, W_m]$
$e_{ij} = \mathrm{dot}(F(a_i)^T, F(b_j))$
E matrix
W_1 W_2 W_3 … W_n
W_1
W_2
W_3
…
W_m
e_{ij}
F feed forward
300
200
200
$\boldsymbol{\beta}_i = \sum_{j=1}^{l_b} \frac{exp(e_{ij})}{\sum_{k=1}^{l_b} exp(e_{ik})} b_j$
$\boldsymbol{\alpha}_i = \sum_{i=1}^{l_b} \frac{exp(e_{ij})}{\sum_{k=1}^{l_a} exp(e_{kj})} a_j$
$V_{1,i} = G([a_i, \beta_i]) \forall i \in [1, \dots, l_a]$
$V_{2,j} = G([b_j, \alpha_j]) \forall j \in [1, \dots, l_b]$
(Where [*,*] is concatenation)
G feed forward
400
200
200
$V_1 = \sum_{i=1}^{l_a} V_{1,i}$
$V_2 = \sum_{j=1}^{l_b} V_{2,j}$
$\hat{y} = H([V_1, V_2])$
(Where [*,*] is concatenation)
H feed forward
400
200
3

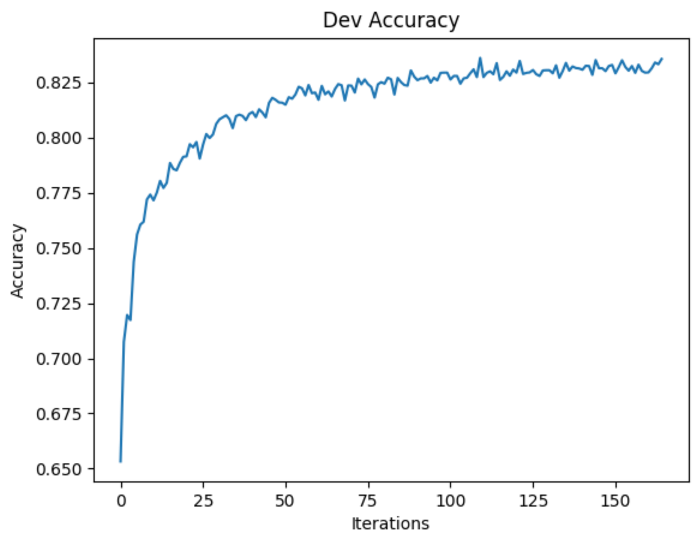
- Did your code manage to replicate this result?

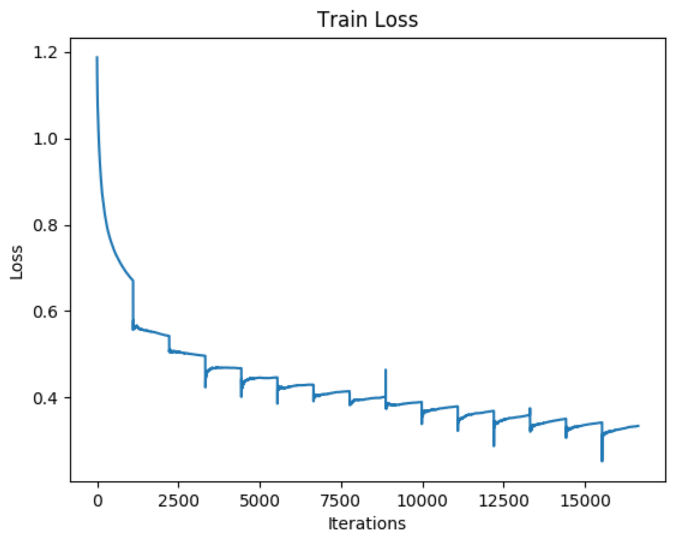
No, it did not.

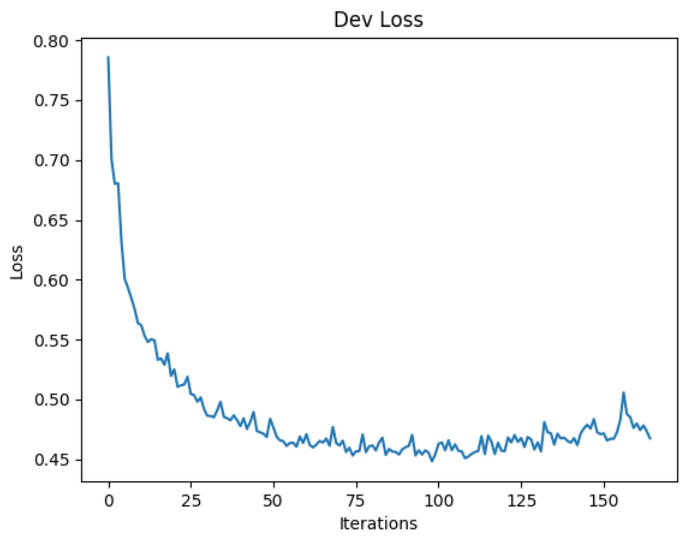
- What was your performance on that dataset (how does your report compare to theirs)?

	Theirs		**Ours**		**Δ**	
	Train	Test	Train	Test	Train	Test
Accuracy	**89.5%**	**86.3%**	**86.2%**	**83.1%**	**-3.3%**	**-3.2**

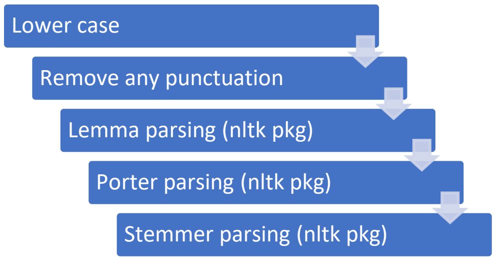








- <u>What was involved in replicating the result?</u>
  - One of the biggest challenges we had, was to find the words in the GloVe embeddings data. We started our model with direct tries of getting the words out of the data, but we found out that we get a lot of UNKs. We did some parsing jobs for cutting down those random UNK embedding and get the significant word embeddings. We tried several ways and decided on the attached one. We checked if the word appears in the embedding data, if it doesn't, we did some parse and then check again. After all those parses the only words that which didn't have embeddings were the words with spelling mistakes, we tried to do some basic job on that, and found out that the spelling mistakes is a very interesting problem by itself. Each UNK word was mapped to some random embedding and saved to a dictionary, thus the same UNK word got every time the same embedding.



  - We tried to change the size of the network (several configurations), the activation functions (to tanh), number of 'UNK' embeddings, but find out that the best results are achieved with the paper's hyperparameters.
  - We did change the optimizer to Adam that achieved a better accuracy.
  - We did change the learning rate (that was 0.05 at the paper) to 0.001 . We did it because we had vanishing gradients issues. That probably was because of the difference between the platforms TensorFlow (paper) vs. DyNet (ours).
  - We did change the batch size to 16, thing that helped for some speed up in our model.
  - We ran our model for 15 epochs.

- <u>What worked straightforward out of the box? what didn't work?</u>

Coding the model with DyNet was quite simple. That was the first time that we did such complexed model that has several feed-forward networks and attention in it. We were not sure how to forward and backward through the whole model, but DyNet provides its tools to make the workflow easy.

The difficulties we had were: loading the embedding data and parse each word to be found in the embedding.

- <u>Are there any improvements to the algorithm you can think about?</u>
  - Pay attention to the word location in the sentence (a thing that is discussed in that paper as optional).
  - Devote more attention to the spelling mistakes problem that will easily improve the accuracy, since more words will be appear in the embeddings.
  - Add unknown words to the embeddings and train them.